AUGUST 2014

FM-BIASED

Formal Methods: Business Impact of Application to Security relevant Devices

Applying the Scyther formal verification tool on the DLMS/COSEM standard – Version 2.1

FM-BIASED
Formal Methods: Business Impact of Application to Security relevant Devices

Applying the Scyther formal verification tool on the DLMS/COSEM standard - 2014

Authors: Sara Cleemput, prof. Geert Deconinck, Yoni De Mulder, Kristof Devos, prof. Bart Preneel, Stefaan Seys, Dave Singelée, Alan Szepieniec, Pieter Vingerhoets

With the financial support of the Prevention, Preparedness and Consequence Management of Terrorism and other Security-related Risks Programme European Commission - Directorate-General Home Affairs
The Commission is not responsible for any use that may be made of the information contained therein, the sole responsibility lies with the authors.

ABSTRACT

This report discusses the application of the Scyther formal verification tool on the DLMS/COSEM standard for smart metering, a case study in the scope of the FM-biased project. This case study was carried out at the Leuven University in Belgium. Smart meters send fine-grained consumption data to the utility without any manual intervention. This allows the utility to carry out fine-grained billing and manage the distribution grid more effectively. By 2020 at least 80% of the European consumers will be equipped with smart metering systems, according to EU directive 2009/72/EC.

However, this automated metering communication can potentially lead to serious security threats. If malicious entities are able to intercept meter data, they can infer private information about the consumer, e.g. learn about his daily routine. If they are able to send falsified metering data, they can commit fraud, by not paying for the energy they consumed. The potentially largest impact is that in the latter case, they can also influence the grid management carried out by the Distribution System Operator and cause major black-outs. To avoid this, the DLMS/COSEM standard includes authentication mechanisms for meters. In this report, we use the formal verification tool Scyther to show that this authentication mechanism can be circumvented. We also compare the Scyther verification tool to manual methods for security analysis.

Summary

ABSTRACT

1. Introduction 6
   1.1 The smart metering case 6
   1.2 The DLMS/COSEM standard 7
   1.3 The Scyther tool 9
2. Results of the formal verification 13
   2.1 Applying Scyther 13
   2.2 Practicality of the attack 18
   2.3 Impact of the attack 19
3. Comparison of formal and manual methods 22
   3.1 Manual verification 22
   3.2 Scope 24
   3.3 Cost and usability 26
4. Conclusion 27
5. REFERENCES 28

Summary of figures

Figure 1: The DLMS/COSEM layered communication model [3] 8
Figure 2: The High Level Security protocol [2] 9
Figure 3: The Scyther formal verification model [4] 12
Figure 4: The HLS protocol description in Security Protocol Description Language 14
Figure 5: The verification of the claims by Scyther 14
Figure 6: The attack on the client found by Scyther 16
Figure 7: The attack on the server found by Scyther 17
Figure 8: Status of the smart meter roll-out in Europe [9] 20

Summary of tables

Table 1: Number of nstalled smart meters in Europe by 2020 [7] 21

1. Introduction

1.1 The smart metering case

In the traditional electricity grid, electricity is generated centrally, using fossil fuels. This energy is then transported unidirectionally from the producer to the consumer via the transmission and distribution grid. The amount of energy produced and consumed should be equal at all times, since storing large amounts of energy is very inefficient. In the traditional model, the generation is adapted to the consumption. This means the generation must be predictable and flexible.

However, this model of centrally generating electricity using fossil fuels is not sustainable anymore. Climate directives indicate that by 2020, 20% of the electricity consumption should originate from renewable sources. Thus, more and more renewable energy sources, like wind and solar energy are added to the grid. These often generate much smaller amounts of energy and are therefore also called Distributed Energy Resources (DERs). Consequently, in this new model, energy is produced and consumed locally, flowing both up- and downstream. Moreover, since these DERs consist mainly of photovoltaic cells and wind mills, they are very unpredictable. Therefore, the consumption of electricity needs to be adapted to the production, a concept known as demand-response. To make this possible, the smart electricity grid [1] was introduced.

One of the most important components in this smart grid is the smart meter. This smart meter communicates with the utility in an automated and bidirectional manner, allowing the utility to receive the consumption data of each household multiple times per hour. This allows the Distribution System Operator (DSO) to manage the grid much more efficiently, since it has a near real time overview of the state of the grid. It also allows demand-response, since the consumption can be influenced by fine-grained billing.

The disadvantage however, is that this remote metering is susceptible to impersonation attacks. A malicious entity can pretend to be the smart meter and send false consumption data to the utility. This can be done in order to commit fraud, i.e., to avoid paying for the energy consumed. Much more alarmingly, it can also be done to influence the grid management decisions taken by the DSO. If an attacker can impersonate several meters in one neighbourhood and report consumption values which are much

higher or lower than the actual values, he can lead the DSO to believe that there is over- or undervoltage on the grid, while in reality there is not. The (unnecessary) corrective actions then taken by the DSO can lead to actual voltage-problems on the grid. An attacker pretending to be the utility on the other hand can discover the consumption patterns of many households, thereby learning various sensitive information, concerning for example religion or health.

Finally, smart meters can also be used to measure gas or water consumption. In this case, the smart gas and water meters usually communicate with the communication module of the smart electricity meter. This communication module then sends the consumption data for all three modalities to the utility.

1.2 The DLMS/COSEM standard

DLMS/COSEM stands for Device Language Message Specification, Companion Specification for Energy Metering. This standard is published as IEC 62056-5-3 [2]. It specifies the interface model and communication protocols for data exchange between the smart meter and the utility. DLMS/COSEM is not implementation specific, meters by different manufacturers can all be DLMS/COSEM compliant.

The DLMS/COSEM communication model [3] follows a layered approach, as is illustrated in Figure 1. The top layer is the application layer, which forms the logical connection between the smart meter and the utility. This is the only COSEM specific layer. The lower layers transport the information. These underlying layers are not specified by the standard, meaning that any combination of layers capable of providing the services required by the application layer can be used. A specific set of layers capable of executing the COSEM application processes, is called a communication profile. Examples of communication profiles are: HDLC over a physical layer (a local optical port, the GSM network …) providing connection-oriented asynchronous data exchange; or TCP-UDP/IP over a physical layer (Ethernet, GPRS …), supporting data exchange via the Internet.

DLMS/COSEM employs a client-server model. The server is always the smart meter communication module; the client is the back-office, for example the SCADA system of the DSO. Usually the client initiates the communication, for example by requesting meter data or sending a command. However, in exceptional cases the server can also initiate communication, for example to send out an urgent alarm. A client can communicate with different servers concurrently and vice versa.

Communication between client and server takes place by exchanging messages, SERVICE.requests and SERVICE.responses. DLMS is a connection-oriented protocol, meaning that client and server need to be associated before they can start to communicate. This means that every communication consist of three phases: connection establishment, data transfer and connection release.

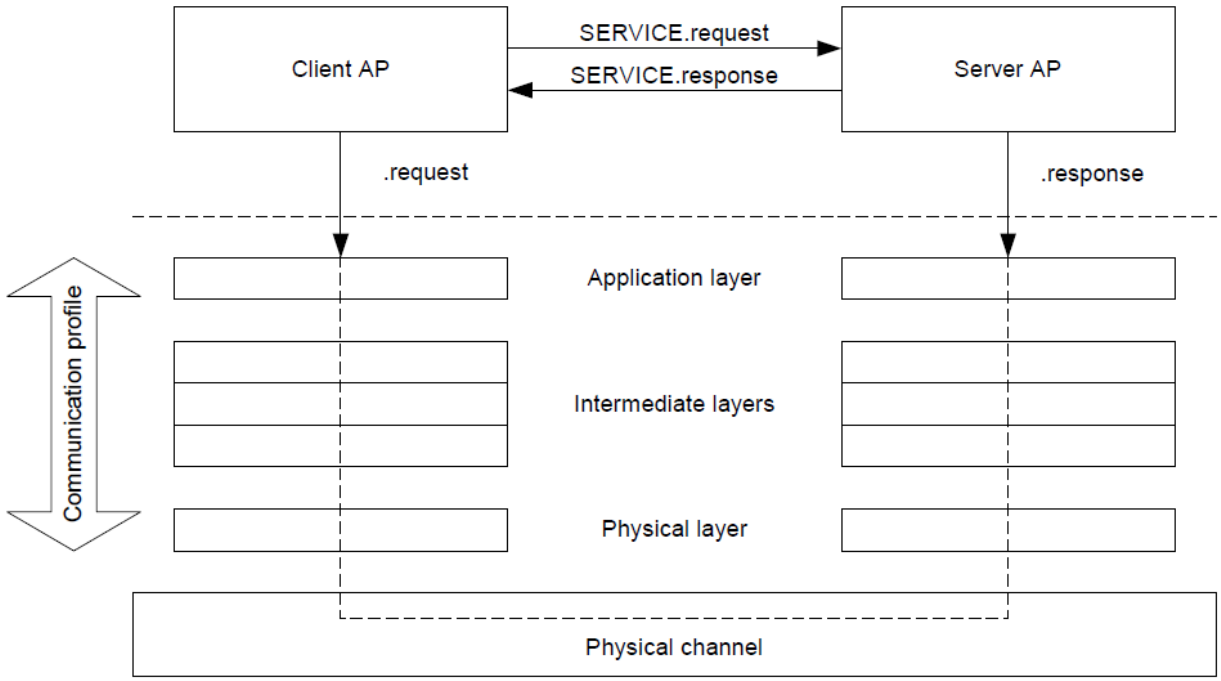

Figure 1: The DLMS/COSEM layered communication model [3]

The DLMS/COSEM standard defines three levels of security [2] for authentication during the connection establishment: lowest level security (no security), Low Level Security (LLS) and High Level Security (HLS). Lowest level security is only used to retrieve very basic information. LLS consist of sending the client password in plaintext, the server is not authenticated. This mode should only be used if the channel can be assumed completely secure against eavesdropping. In this report we consider the HLS protocol, where mutual authentication is attempted by means of a challenge-response protocol. This protocol is shown in Figure 2. It is initiated by the client, i.e. the back-office, sending a challenge (*CtoS*) to the server, i.e. the smart meter. The server also sends a challenge (*StoC*) back to the client. The client then calculates the response to the challenge it received from the server (*f(StoC)*) and sends this response to the server. The function "*f()*" depends on the secret key C_K, so by sending *f(StoC)* to the server, the client proves that it has knowledge about this secret key and thus is an authorised node in the network. The server checks

whether this response is what it expected and sends back the response to the challenge it received from the client (*f(CtoS)*), in turn proving that he also has knowledge of the secret key C_K. Finally, the client also checks the response it received. If either party receives an incorrect response, it will break off the communication. If both responses are correct, the data transfer phase can proceed.

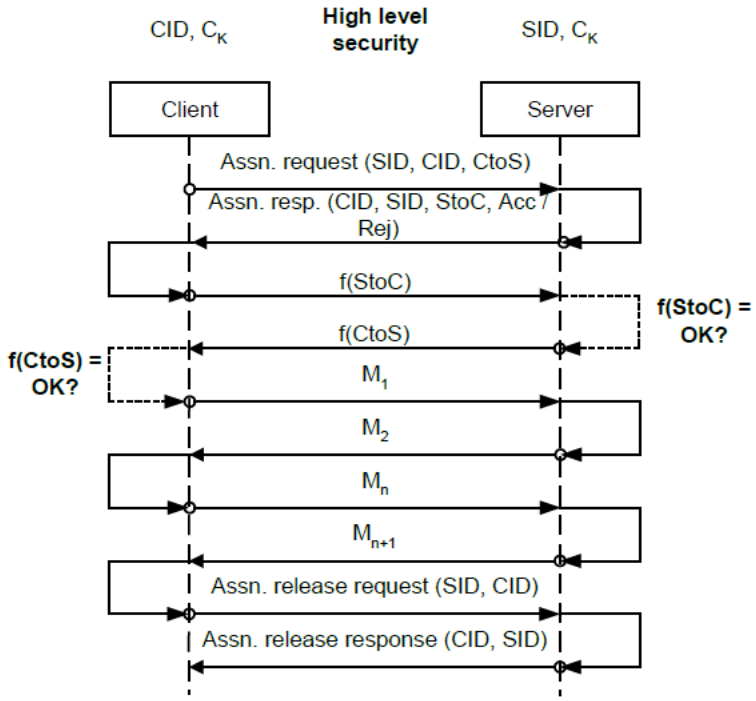

Figure 2: The High Level Security protocol [2]

The DLMS/COSEM standard also defines four policies for data transport security [2]: security is not imposed, all messages are authenticated, all messages are encrypted, all messages are authenticated and encrypted. In this report we consider the policy where all messages are authenticated, as described above.

1.3 The Scyther tool

The Scyther tool [4], [5] is a formal verification tool for security protocols, developed by prof. Dr. Cas Creemers. The main advantages of Scyther are that It can verify protocols with an unlimited number of sessions and nonces, it can give a finite representation of all possible protocol behaviours and it is efficient. The Scyther tool can also analyse the composition of multiple protocols executed in parallel. The

use of Scyther led to the discovery of several new attacks, illustrating that composing individually secure protocols does not necessarily lead to a secure protocol.

Scyther is based [6] on the black-box model, defined by Dolev and Yao. This black-box model consists of three assumptions: firstly, it assumes that all cryptographic functions (for example encryption, hash functions) are perfect. Secondly, messages are defined as abstract terms: the adversary learns either nothing or the enitre message. Thirdly, the adversary is assumed to have full control over the network. Hence every sent message is immediately observed by the intruder, and any received message is supplied by the intruder. We assume the intruder can generate any number of constants of the type of each variable, and also possesses a finite set of terms stemming from the compromised agents with whom he conspires. This includes the private keys of these agents, and as a result the intruder can impersonate these agents.

This underlying model is expressed in terms of operational semantics, which allows us to formally define protocol behaviour and security claims. These security claims are the security properties the user wants to verify. The Scyther tool will look for an event trace that ultimately will break one of these security claims. To this end the Scyther algorithms implement model checking with respect to the unbounded model performing a backward-style search. For these methods, the model is extended with adversary events for encrypting, decrypting, hashing, and knowing messages. During backward search, a case distinction on the source of messages is used for branching and the patterns are extended by either adding events, adding ordering constraints, or unifying terms.

The search can terminate in two ways. Firstly, the pattern can be proven to be empty, i.e., it contains no traces of the protocol. The main mechanism here is detecting cyclic dependencies of the messages. Secondly, if the receive events in the pattern meet all premises of the receive event: the adversary can produce an appropriate message from the preceding events. In such a case, the pattern is called realizable and it corresponds to an infinite set of actual traces; a representative trace (of minimal length) from this set can be generated by linearizing the non-adversary events and instantiating the remaining variables from the adversary knowledge.

By bounding the size of the patterns, termination is guaranteed and one of three possible results occurs. Firstly, if a realizable pattern is found, a representative (attack) trace is constructed. Secondly, if no realizable patterns are found, and the bound is not reached, no realizable patterns exist (for any bound).

In case of an attack pattern, this corresponds to the absence of attacks. Thirdly, if no realizable patterns are found but the bound is reached, the result can be interpreted as verification with respect to a bounded number of sessions.

Binary distributions of Scyther are available for Linux, Windows and OS X. Scyther is still under active development and has a GUI, built on Python, that can be used to visualise attack traces and the validity of security claims. The documentation is rather extensive and a large set of examples is provided as a tutorial to get users started. For the protocol description file, no assistance is provided, so the user will have to draft the protocol description (ASCII text file) himself, and any mistakes will only be known when starting the analysis.

A screenshot of the Scyther verification tool is shown in Figure 3. The left window shows the input to the tool: a translation of the protocol into Security Protocol Description Language (.spdl), including the security claims. Protocols are described as a set of roles, which in turn are described by a sequence of events. These events mainly consist of the sending and receiving of messages and security claim events. The messages can contain constants, nonces, encrypted values and hash values. Examples of security claims are: secrecy of a term, aliveness of all roles, non-injective synchronisation etc. The middle window in Figure 3 displays the results of the claim verification. For every claim that fails to be verified, at least one attack is possible. The rightmost window illustrates such an attack. Section 2.1 explains this process in more detail, for the DLMS/COSEM HLS protocol.

Scyther has been used to analyse the IKEv1 and IKEv2 protocol suites and the ISO/IEC 9798 family of authentication protocols. The tool has also been used to find new multi-protocol attacks on many existing protocols.The tool comes bundled with a set of example protocol files modeled from the SPORE repository, which is set up to be an online continuation of the Clark-Jacob library of authentication protocols. The compromising adversaries version of Scyther has been applied to a large amount of Authenticated Key Exchange (AKE) protocols, such as HMQV, UM, NAXOS, and many other protocols.

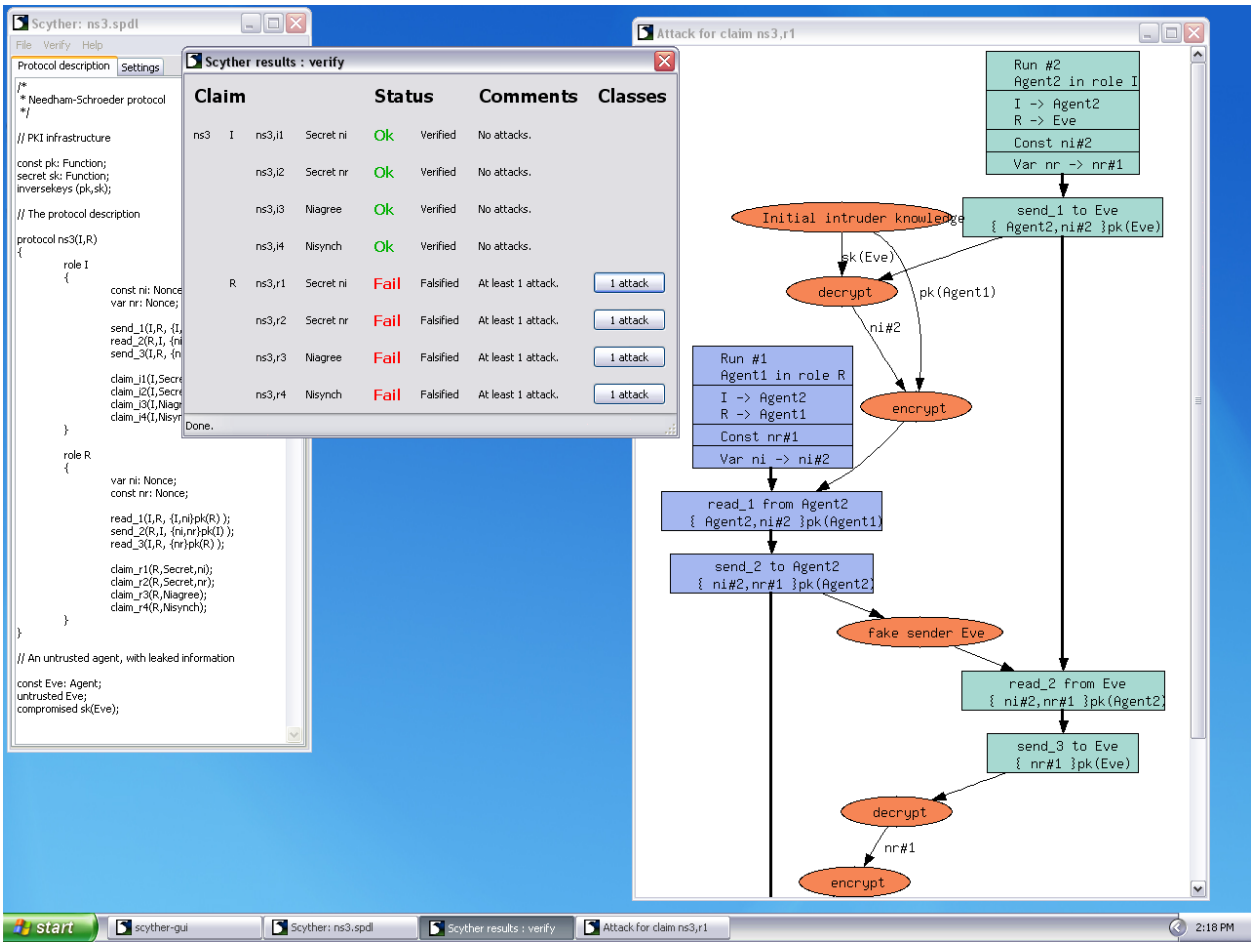

Figure 3: The Scyther formal verification model [4]

2. Results of the formal verification

2.1 Applying Scyther

We apply the Scyther formal verification tool [4] to the DLMS/COSEM High Level Security protocol for data access. The first step is in this process is converting the protocol into Security Protocol Description Language [5], which is illustrated in **Error! Reference source not found.**. There are two roles: the client and server. Both first need to generate their challenge (*CtoS* and *StoC*). This challenge should be a freshly generated number that will only be used once (a so called nonce). The first event then consists of the client sending its challenge to the server, followed by the server sending its challenge to the client. The third event consist of the client sending the response to the challenge it received from the server. We assume this response is calculated as the symmetric encryption of the challenge. Indeed this is a good response function, since the encryption depends on the shared secret key. The only entities with access to this key are the server and the client, so no third party is able to calculate the correct response. Finally the server sends to the client the response to the challenge it received from him. Again this response is calculated as the symmetric encryption of the challenge.

Now that the protocol flow has been defined, we need to define the security claims. For both of the roles we claim non-injective synchronisation [6]. Synchronisation means that all messages sent by one party are indeed received by the other party and vice versa. In other words, both parties know exactly who they are communicating with and they agree on the contents of the messages that have been sent. Two things should be noted here: if an adversary intercepts the messages, inspects their contents and then simply forwards them to the intended recipients without altering the contents, this does not violate the synchronisation property. Secondly, protocols that have the non-injective synchronisation property may still be vulnerable to replay attacks. A replay attack is an attack where a malicious entity resends an earlier trace to one of the honest communication partners, leading him to believe he is communicating to another honest partner.

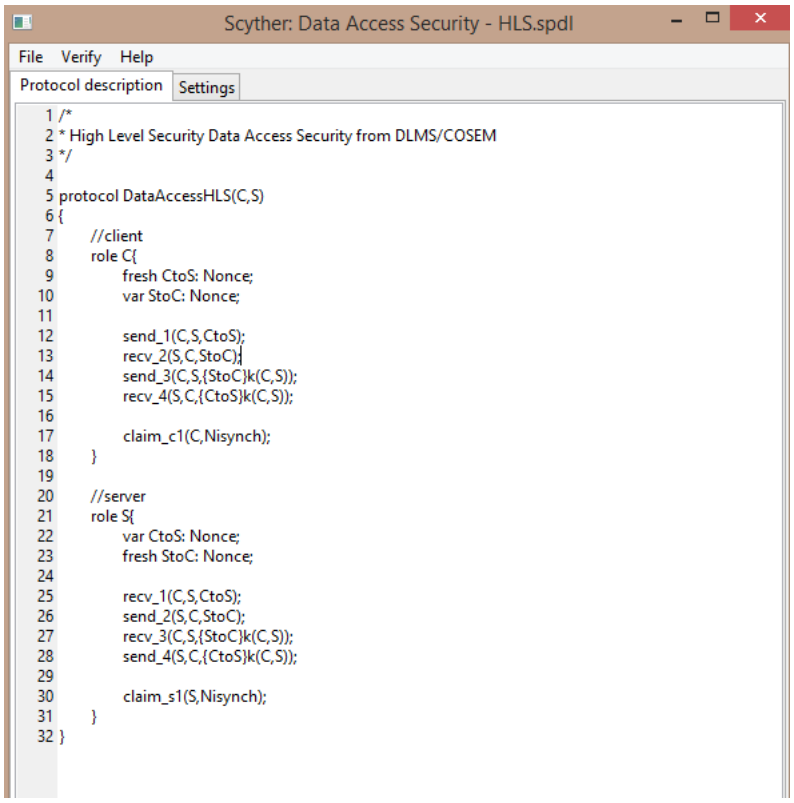

Figure 4: The HLS protocol description in Security Protocol Description Language

When running the verification, the Scyther tool shows that the synchronisation claim is invalid for either role, as is shown in Figure 5. This means that the DLMS/COSEM HLS protocol is not a secure protocol.

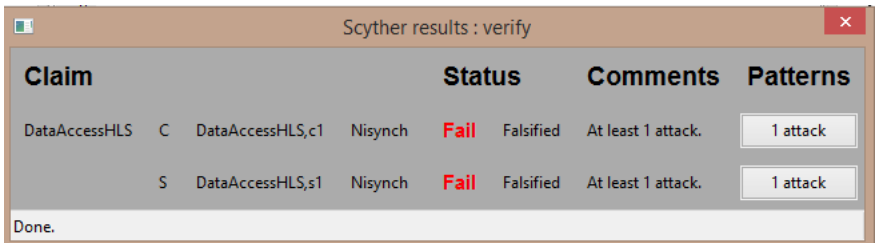
Scyther results : verify
Claim
Status
Comments
Patterns
DataAccessHLS C DataAccessHLS,c1 Nisynch Fail Falsified At least 1 attack. 1 attack
S DataAccessHLS,s1 Nisynch Fail Falsified At least 1 attack. 1 attack
Done.

Figure 5: The verification of the claims by Scyther

The attack on the client found by the Scyther tool is shown in Figure 6. This attack proceeds as follows: in a first step the client generates a fresh challenge, *CtoS#1* and sends it to the server. This is shown in the top two green boxes in Figure 6. This is exactly as expected from the protocol. However, a malicious third party intercepts this challenge and sends the exact same value back to the client, as shown in the top orange box. The client will assume this is the challenge it receives from the server, as is shown in the third green box. In other words, the client thinks it receives *StoC* from the honest server, but in reality receives *CtoS#1* from the malicious third party. The client then proceeds with the protocol by sending the response (i.e., the symmetric encryption of the challenge it just received) back to the server, as can be seen in the fourth green box. However, the malicious party again intercepts this message, as shown in the bottom orange box. Finally, the malicious entity sends this exact same response back to the client, as is shown in the bottom green box. As the use of symmetric encryption implies that the same challenge will always lead to the same response, the client will accept the response it receives from the malicious entity as the correct one and will assume it is communicating with the honest server.

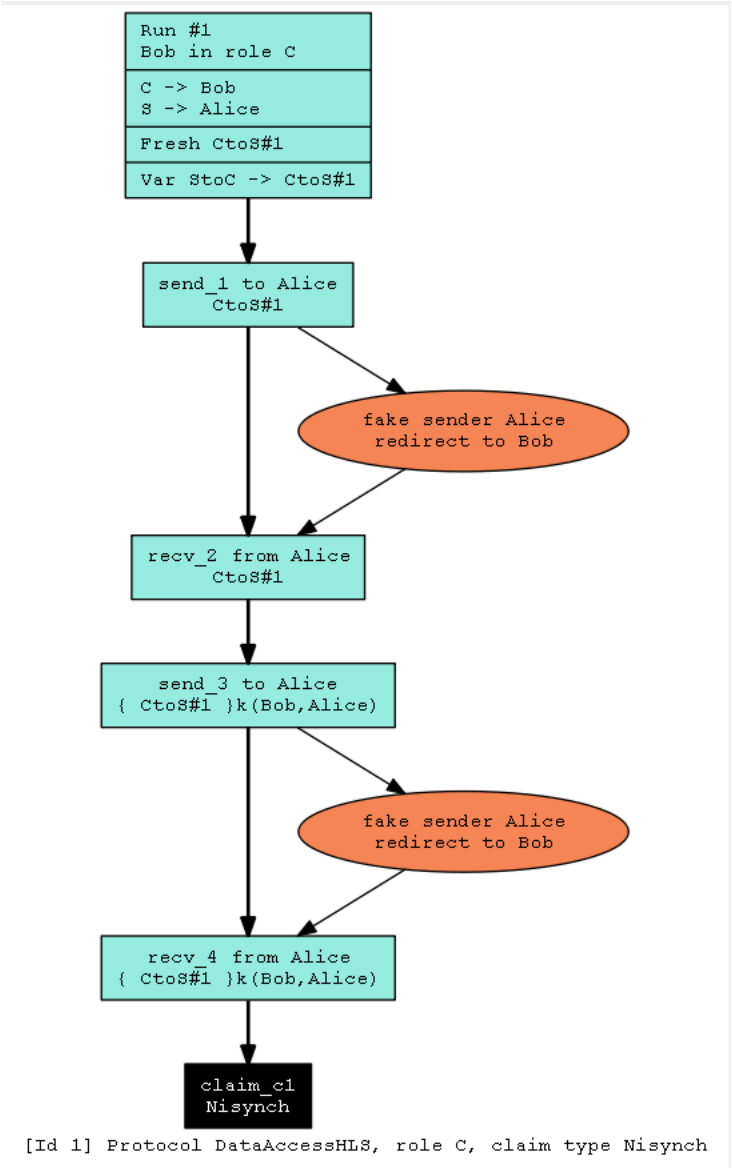

Figure 6: The attack on the client found by Scyther

The attack on the server, shown in Figure 7, is a more complicated attack. In this attack two runs of the HLS protocol are executing in parallel, one between the client and the malicious entity (run #2) and one between the server and the malicious entity (run #1). In the first step the honest client initiates the second run of the protocol by sending its challenge, *CtoS#2*, to the server. The malicious entity however, intercepts this challenge and intiates the first run of the protocol by sending its own challenge, *NonceIntruder1* on to the server. The server thinks that it received a genuine challenge from an honest client and sends its own challenge *StoC#1* back to the client. The client receives this challenge and sends

its response, the encryption of *StoC#1*, back to the server. Finally the server will send the reponse to the challenge it received, i.e. the encryption of *NonIntruder1* back to the client, but the malicious entitiy will again intercept this message. At this point the first run of the protocol is completed, and the server will think it is communicating with a honest client, but in reality is is communicating with a malicious third party. This type of attack is known as a man-in-the-middle attack. The second run of the protocol will never finish, since the client will never receive a response to its challenge, which was dropped by the malicious entity.

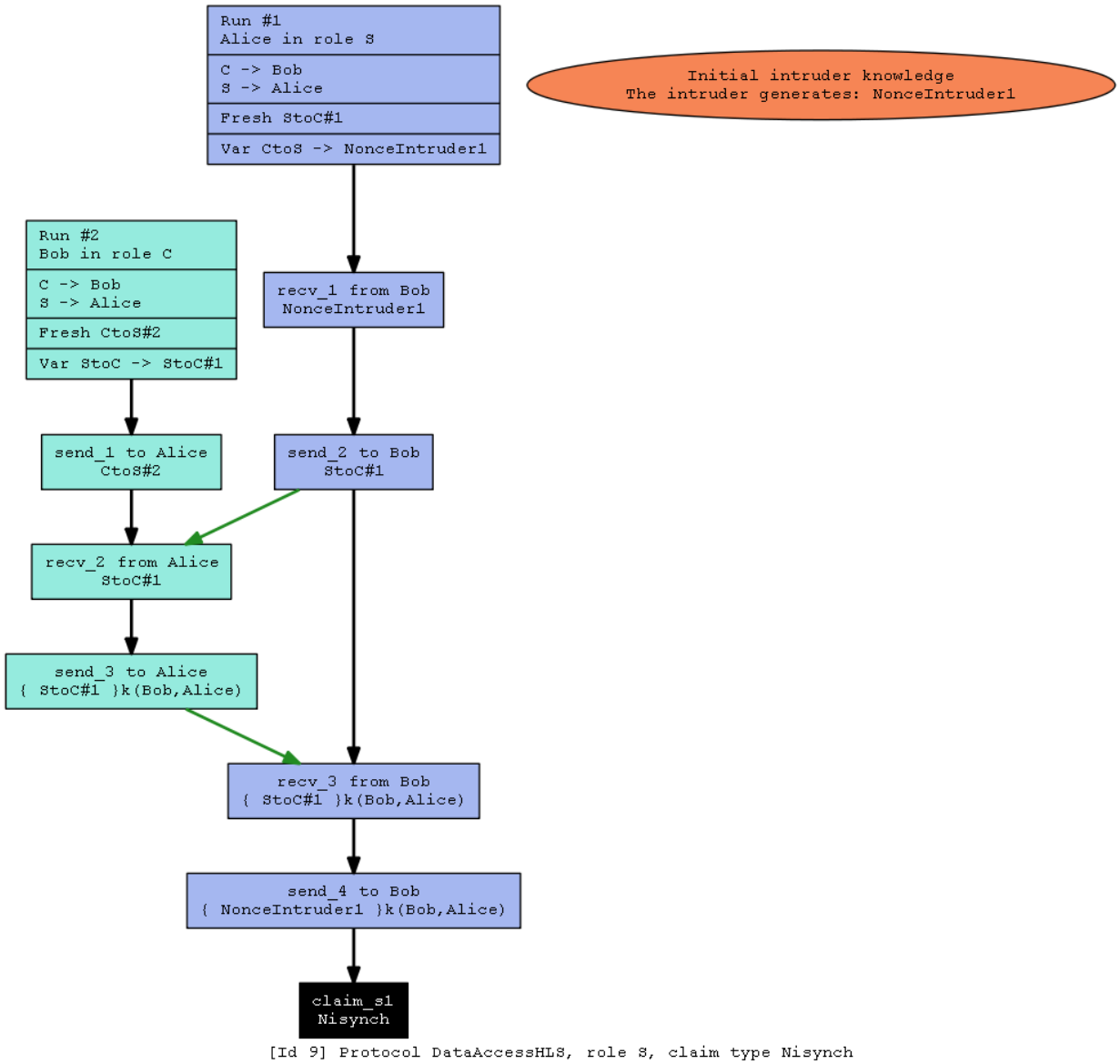

Figure 7: The attack on the server found by Scyther

2.2 Practicality of the attack

In order to describe how this attack would be carried out in practice, we again remind the reader that in the DLMS/COSEM model the server represents the smart meter and the client represents the utility. This means that the attack on the client, shown in Figure 6, corresponds to a malicious party, pretending to be a smart meter, communicating to the honest utility. The attack on the server, shown in Figure 7, corresponds to a malicious third party, pretending to be a genuine utility, communicating to an honest smart meter.

A very easy to implement and practical attack scenario is the following. Every household will have a smart meter that is connected to the utility. Normally, a smart meter is authenticated using the authentication protocol we have described above. As this protocol is insecure, any person with a smart meter will be able to replace this meter with a fake device that spoofs fake meter data. If the communication system between the utility and the different houses is not point-to-point, but employes a shared media (e.g., a LAN-like network), then this attack can be extended to spoof all meters that are part of this broadcast domain. In this case, one person could potentially spoof meter data of complete streets or neighbourhoods.

In a second, less practical attack scenario, a malicious third party could try to pretend to be the genuine utility in order to read out all meter data, change settings on the smart meters, etc. In order to execute this attack, the attacker has to place a device “between” the genuine utility and the genuine smart meter that he wants to target. When the smart meter is communicating wireless, this would involve complex hardware to block signals while still receiving them, etc. If the communication is wired, then the attacker will need physical access to the cable running from the target smart meter.

Both attacks require an attacker to possess knowledge about the DLMS/COSEM protocol and the underlying layers. He needs to understand the correct format and structure of the messages exchanged, as well as how they should be presented to the lower layers. The attacker also needs to have physical access to communication channel. In the case of wireless communication this is trivial, for wired communication more effort might be required from the attacker. It should also be noted that both

attacks can only be executed when an honest utility initiates the communication, since both attacks require a genuine challenge from the client client (i.e., the central systems of the utility).

2.3 Impact of the attack

According to the third Energy Package, in 2009 the member states in Europe have conducted a cost-benefit analysis for the roll out of smart meters by 2012. In case of a positive cost-benefit analysis, regulation requires a large-scale roll-out with 80% of the customers being equipped with smart meters by 2020. An overview of the results of this analysis in the different member states is shown in Figure 8. Currently, several millions of smart meters have already been installed in Europe, as illustrated in a recent report issued by the European commission [7]. By 2020 there will be an estimated 200 million smart meters deployed in Europe. This clearly indicates that the potential consequences of a security breach will be substantial.

However, if users can commit fraud by reducing their energy bill by only 1%, this still corresponds to 24TWh in total. In current electricity prices (20c/kWh in 2013, [8]), this corresponds to a loss of approximately € 4.8 billion. These numbers clearly indicate that it is of the utmost importance that protocols used to transfer smart metering data are secure.

Another type of problem is related to privacy. According to European regulation all data which can relate back to a person are subject to strict rules. Consumption data reveal a lot of information about the consumer. Not only the type of appliances the consumer owns and his consumption pattern, but also lifestyle information and other sensitive information, such as religious beliefs and health information. Consequently, compensation claims of millions of euros could be made, with significant financial consequences for the companies involved.

The third and most serious problem arises when consumption information is used to maintain the grid voltage within acceptable limits. Currently, active demand scenarios with voltage control are still in the demonstration phase and are not yet used on a large scale. However, in future implementations of the grid, it is of paramount importance to ensure that communication errors will not result in voltage problems and/or outages.

In conclusion, smart meter security is a vital issue that must be taken into account at all stages of the design life cycle. Failures in the DLMS-COSEM protocol like those found by the Scyther tool are unacceptable and could lead to significant financial damage for both consumer and DSO. A security check performed manually and/or using formal methods is necessary, before publishing an update of the protocol.

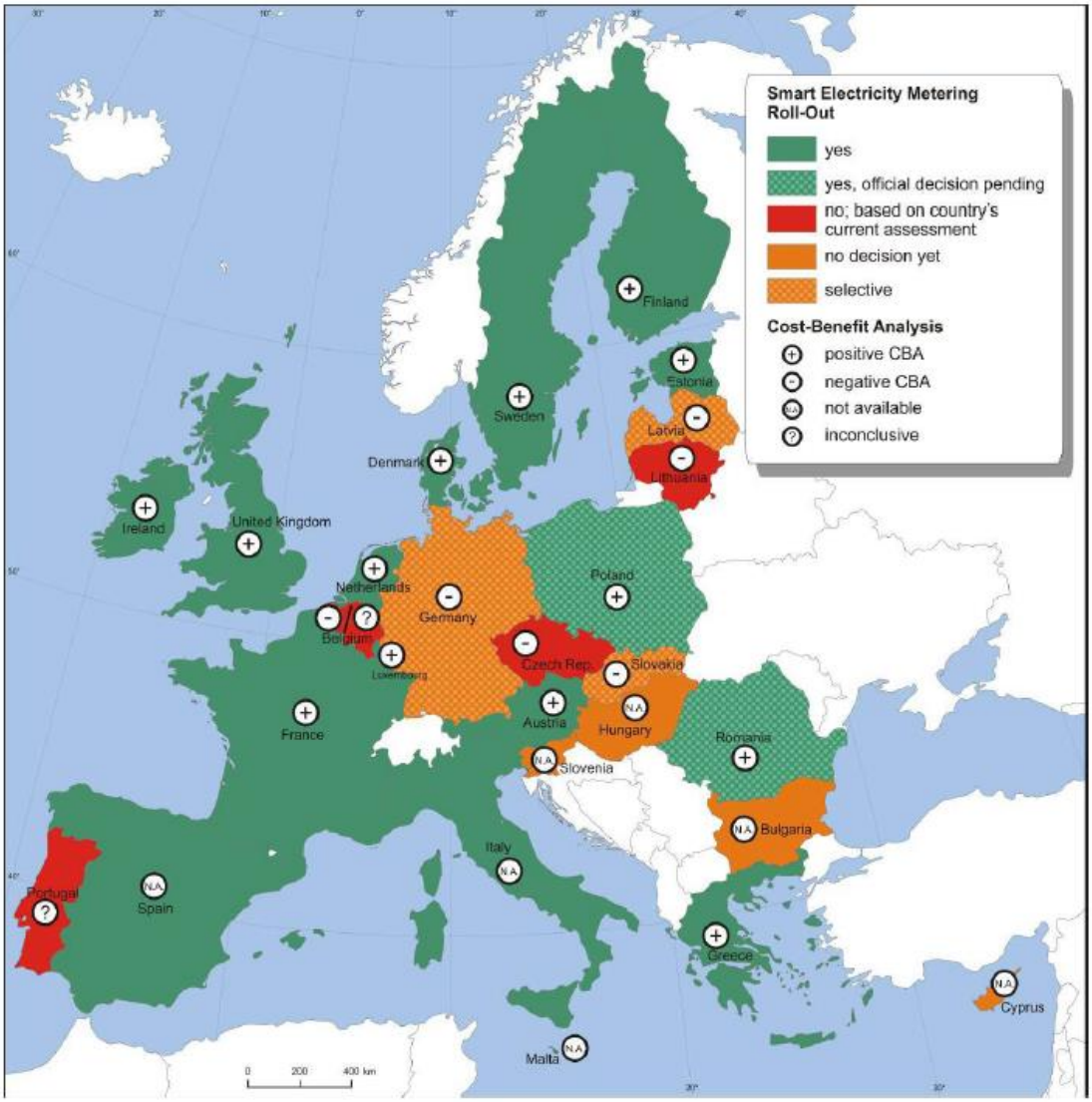

Figure 8: Status of the smart meter roll-out in Europe [9]

Large-scale roll-out (at least 80% of consumers by 2020)	Metering points in the Country by 2020	Expected Diffusion rate by 2020 (%)	Total Number of Smart Metering Points to be installed up to 2020
Austria	5700000	95%	5415000
Denmark	3280000	100%	3280000
Estonia	709000	100%	709000
Finland	3300000	100%	3300000
France	35000000	95%	33250000
Greece	7000000	80%	5600000
Ireland	2200000	100%	2200000
Italy	36700000	99%	36333000
Luxembourg	260000	95%	247000
Malta	260000	100%	260000
Netherlands	7600000	100%	7600000
Poland	16500000	80%	13200000
Romania	9000000	80%	7200000
Spain	27768258	100%	27768258
Sweden	5200000	100%	5200000
United Kingdom — GB	31992000	99.5%	31832040
Total	**192469258**	**95.3%**	**183394298**

No large-scale roll-out (at least 80% of consumers by 2020)	Electricity Metering points in the Country by 2020	Expected Diffusion rate by 2020 (%)	Total Number of Smart Metering Points to be installed up to 2020
Belgium	5975000	NA	NA
Czech Republic	5700000	1.0%	57000
Germany	47900000	23.0%	11017000
Latvia	1089109	23.0%	250495
Lithuania	1600000	NA	NA
Portugal	6500000	NA	NA
Slovak Republic	2625000	23.0%	603750
Total	**71389109**	**16.7%**	**11928245**

Table 1: Number of nstalled smart meters in Europe by 2020 [7]

3. Comparison of formal and manual methods

3.1 Manual verification

To analyse the security properties of an ICT system, threat modeling and risk assessment (the combination of both is denoted by 'risk and threat analysis' in the rest of this document) are the de facto tools to achieve this goal. Indeed, instead of carrying out an ad-hoc security analysis and potentially overlooking some important security issues, a systematic analysis of the system is required. In IT security, threat modeling is used to describe the security issues that should be considered during the design process of a system. Moreover, this method is used to analyse a set of possible attacks on the implementation of the system and the general system architecture. Threat modeling is based on the notion that any system or organization has assets of value worth protecting. These assets have certain vulnerabilities, which can be exploited by internal or external threats to cause damage to these assets. Therefore, threat modeling is typically combined with a study of the system's components, to get a clear overview of the important assets and how valuable they are for an attacker.

Since one can have different views of a system, there is no single method to model the system's threats. More in particular, there are at least three common approaches to threat modeling:

- Attack-Centric: Attack-centric threat modeling starts with an attacker, and evaluates his/her goals, and how he/she might achieve them. I.e. the designer views the system from the viewpoint of an attacker to determine the threats. This approach usually starts from either entry points or assets.
- Software-Centric: This approach starts from the design of the system and attempts to step through a model of the system, looking for types of attacks against each element of the model. Therefore, this approach requires that a system model is constructed in advance, where all the system's components can be identified.
- Asset-Centric: The asset-centric threat modeling approach starts from the assets and looks for attacks against each of these assets.

Besides these three common methodologies, there are also hybrid threat modeling techniques which combine various aspects of attack centric, software centric and asset centric threat modeling.

One of the most commonly used methodologies to classify threats in an IT system is STRIDE. This is a software-centric threat modeling methodology developed by Microsoft, and among others used in Microsoft's Security Development Lifecycle. STRIDE is an abbreviation of six threat categories:

- Spoofing: A spoofing attack is a situation in which one user or program successfully masquerades as another by falsifying data and thereby gaining an illegitimate advantage.
- Tampering: Data tampering involves the malicious modification of data.
- Repudiation: Repudiation threats are associated with users who deny performing an action (e.g., an illegal operation) without other parties having any way to prove otherwise.
- Information disclosure: Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it.
- Denial of service: Denial of service (DoS) attacks deny service to valid users. For example, by making a certain service unavailable.
- Elevation of privilege: In this type of threat, an unprivileged user gains privileged access, which could potentially result in a compromise or destruction of the system.

Although the three approaches to threat modeling, including the STRIDE methodology which will be applied in this project, were originally defined to analyse the security of software systems, they can be used more broadly. For example, software-centric threat modeling can be applied on each system that can be illustrated by means of data-flow, use case or component diagrams. This includes smart meter systems, the context of this project.

However, the threat modeling methodology, such as STRIDE, cannot be applied directly to each system, since it may not fit all organizations and systems. For example, the threat category 'elevation of privilege' used in Microsoft's STRIDE methodology is probably not very relevant for smart grids and can therefore be omitted, but other threat categories (such as privacy threats) could be very useful to include. Fortunately, the STRIDE methodology can easily be fine-tuned to match the system that needs to be analysed. It is also important to note that each software-centric threat modeling methodology heavily depends on the details of the system diagram: components and interfaces which are not described in this diagram will not be analyzed. Therefore, it is important to have a complete diagram of the system.

The result of applying the STRIDE methodology to the smart meter system will be a table with potential security threats. However, not all these threats are evenly relevant or important. Therefore, threat

modeling is typically combined with a risk assessment methodology. In case of the STRIDE methodology, the corresponding risk assessment model is DREAD. It is an abbreviation of five categories to rate the threats:

- Damage: How much damage (e.g., financial loss) is caused by the attack?
- Reproducibility: Is it easy to reproduce the attack, or will it only work once (on a specific (subset of) the system)?
- Exploitability: How much work is needed to execute the attack (e.g., equipment cost, time needed by the attacker, etc.)?
- Affected users: How many users will be affected by the attack? Is it targeted towards one user, or towards a large set of users?
- Discoverability: How easy is it for an attacker to discover the attack? Does it require significant technical skills, or can (almost) everybody execute the attack?

Based on this risk assessment methodology, one can rate the attack according to importance. An attack is more important if it causes significant damage (including loss of reputation), is easy to reproduce, can be exploited without much work or cost, affects a large set of users and/or does not require specialised technical skills. Depending on the specific system that will be analysed (in this case the smart meter system), one could include weight factors for one or more of the DREAD categories. When a given threat is assessed using DREAD, each category is given a rating score (multiplied by the weight factor, if available): e.g., from 5 for very high to 0 for very low. As a result, each threat will get a final score, which can then be used to rank and prioritise the threats. Next, one typically discards all the threats whose score is below a particular threshold. For all other threats, which are the most important and relevant ones, security techniques are being implemented and/or developed to mitigate the threat.

3.2 Scope

The Scyther tool can be applied to any security protocol and can verify the following security claims [6]: secrecy, aliveness, non-injective synchronisation and non-injective agreement. Note that the latter three are all forms of authentication. Secrecy means that the adversary cannot learn the information contained in the message, even though that message is transmitted over an untrusted network and the adversary

can eavesdrop all conversations. This claim assumes that the two communicating partners are honest agents. In other words, if secret information is sent to a compromised entity, thus allowing the adversary to learn the secret information, the secrecy claim is still considered to hold.

An aliveness claim holds if the intended communication partner has executed some event, i.e. is active. Cremers and Mauw mention four forms of aliveness: weak aliveness, weak aliveness in the correct role, recent aliveness and recent aliveness in the correct role. Weak aliveness means the communication partner has executed some event before, during or after the run in which the claim event occurs. Weak aliveness in the correct role extends weak aliveness by also verifying whether the communication partner is executing the role that is expected of it, i.e. initiator or responder. For recent aliveness to hold the communicating partner must executed an event before the claim event, but after some other event from the same run. Finally recent aliveness in the correct role again extends recent aliveness with a verification of the role the communicating partner executes.

Non-injective synchronisation means that all messages sent by one party are indeed received by the other party and vice versa. In other words, both parties know exactly who they are communicating with and they agree on the contents of the messages that have been sent. Two things should be noted here: if an adversary can intercept the messages, inspect their contents and then simply forwards them to the intended recipients without altering the contents, this does not violate the synchronisation property. Secondly, protocols that have the non-injective synchronisation property may still be vulnerable to replay attacks. A replay attack is an attack where a malicious entity resends an earlier trace to one of the honest communication partners, making him believe that he is communicating to another hones partner.

Non-injective agreement on messages means that after execution of the protocol both parties agree on the contents of the variables. The difference between agreement and synchronisation is that agreement does not require events to take place in the correct order. In other words, receive events can take place before their corresponding send events.

In their book Creemers and Mauw also define injective synchronisation and agreement. The added claim of injectivity means that multiple executions of the protocol are all performed as expected, making a replay attack impossible. However, these claims are not available in the Scyther tool.

In general, we can compare the scope of formal methods to that of the manual methods described in Section 3.1. Formal methods can be used to verify cryptographic protocols, but not the security of the

entire system. For example, they cannot check implementation errors, side-channel attacks etc. On the other hand formal verification tools can discover highly complex attack paths, which would probably be overlooked in a manual analysis. The two types of methods are thus complementary to each other.

3.3 Cost and usability

The Scyther tool is freely available. Therefore, the cost can only be expressed in the expertise and time required to apply it. Learning to translate the protocol flow events into the Security Protocol Description Language is not very time-intensive, but understanding the various claims that can be made and interpreting the output of the tool do require a substantial amount of expertise and time. In order to be able to apply the tool correctly one must select the claim corresponding to the security property one is trying to achieve. This is far from trivial and should be done with great care. Not verifying the correct claims can lead one to think there are no attacks on the protocol, while in reality there are. Thus, the Scyther tool can only be used by users with a background in security.

4. Conclusion

In this paper we applied the Scyther formal verification tool on the DLMS/COSEM High Security Level authentication protocol. Using Scyther we found that both the smart meter and the utility can be impersonated when following this protocol. We describe both the attack on the smart meter and the attack on the utility, as well as how to carry out the attack in practice. We also elaborate on the practical impact of such an attack.

The Scyther verification tool is available free of charge, the only cost is the expertise and time required to apply it. Although, protocols can be translated into Security Protocol Description Language relatively easily, extensive security expertise is required to know which claims should be verified and to interpret the output of the tool.

Formal verification methods for cryptographic protocols are certainly an added value compared to applying only manual methods. They can be used to exhaustively verify cryptographic protocols in-depth. However, they are not sufficient in themselves and should be complemented by additional manual analysis, to check for example for implementation errors.

5. REFERENCES

[1] J. G. Slootweg, C. E. P. Jordan Cordova, C. Montes Portela and J. Morren, "Smart Grids - intelligence for sustainable electrical power systems," in *IEEE 33rd International Telecommunications Energy Conference (INTELEC)*, 2011.

[2] *IEC 62.56-5-3: Electricity metering data exchange - The DLMS/COSEM suite - Part 5-3: DLMS/COSEM application layer,* 2006.

[3] DLMS User Association, *DLMS/COSEM architecture and protocols - Excerpt from Companion Specification for Energy metering,* 2009.

[4] C. Cremers, "The Scyther tool," [Online]. Available: http://www.cs.ox.ac.uk/people/cas.cremers/scyther/index.html.

[5] C. Cremers, *Scyther user manual,* 2014.

[6] C. Cremers and S. Mauw, Operational semantics and verification of security protocols, Springer Berlin Heidelberg, 2012.

[7] "ENTSO-E monthly report," [Online]. Available: https://www.entsoe.eu/Documents/Publications/Statistics/2014_Monthly/14_05_MS.pdf.

[8] "European Commission report," 2013. [Online]. Available: http://epp.eurostat.ec.europa.eu/statistics_explained/index.php/Energy_price_statistics.

[9] "European commission staff document," 2013. [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:52014SC0189&from=EN.

FM-BIASED
Formal Methods: Business Impact of Application to Security relevant Devices

Year of publication: 2014
Place: Leuven, Belgium
ISBN:

With the financial support of the Prevention, Preparedness and Consequence Management of Terrorism and other Security-related Risks Programme European Commission - Directorate-General Home Affairs
The Commission is not responsible for any use that may be made of the information contained therein, the sole responsibility lies with the authors.